Backpropagation

$$\begin{tikzcd}
x=1 \arrow[drr, bend left, "\frac{\partial b}{\partial x}=1"] & & & & \\
y=3 \arrow[r, "\frac{\partial a}{\partial y}=-1"] & *(-1) \mid a=-3 \arrow[r, "\frac{\partial b}{\partial a}=1"] & + \mid b=-2 \arrow[r, "\frac{\partial c}{\partial b}=2b=-4"] & (\cdot)^2 \mid c=4 \arrow[r, "\frac{\partial f}{\partial c}=z=2"] & * \mid f=8 \\
z=2 \arrow[urrrr, bend right, "\frac{\partial f}{\partial z}=c=4"'] & & & &
\end{tikzcd}$$

-The algorithm has a runtime of **O(|E|)** since we touch every edge exactly once and requires **O(|V|)** space for the partial derivatives at each node.
-The gradient descent update rule is then: $\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$

Log-Linear Modeling

Exponential family:

$$p(x \mid \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} h(x) \exp(\boldsymbol{\theta} \cdot \boldsymbol{\phi}(x))$$

Definition of a log-linear model:

$$p(y|x, \boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(x, y))$$

We can rewrite the gradient of L as:

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \mathbf{f}(x_n, y_n) - \sum_{n=1}^{N} \mathbb{E}_{Y \sim p(\cdot|x_n, \boldsymbol{\theta})}[\mathbf{f}(x_n, Y)]$$

Therefore, the optimum is where the **expected feature counts** under our model look like the **observed feature counts** from our training data.

$$\text{softmax}(\mathbf{h}, y, T) = \frac{\exp(h_y/T)}{\sum_{y' \in \mathcal{Y}} \exp(h_{y'}/T)}$$

The Hessian is the covariance matrix of the (random) vector $\sum_{i=1}^{n} \mathrm{f}(\mathrm{x}_i, Y)$
-The limit of softmax as T → 0 is the argmax function.
-The limit of softmax as T → ∞ approach uniform categorical distribution (maximum entropy).

Feedforward Neural Networks

Sigmoid	**Hyperbolic tangent**	**ReLU**
$\sigma(x) = \frac{1}{1+\exp(-x)}$	$\tanh(x) = \frac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$	$\mathrm{ReLU}(x) = \max(x, 0)$

Derivatives:

$$\sigma(x)(1-\sigma(x)) \qquad 1 - tanh^2(x)$$

$$ReLU'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$

Each parameter θi get updated:

$$\theta_i \longleftarrow \theta_i - \eta \frac{\partial l(y, \hat{y})}{\partial \theta_i}$$

Each projection layer $W^{(i)}$ lives in $\mathbb{R}^{d_{n+1} \times d_n}$

Example: The final layer lives in $\mathbb{R}^{|\mathcal{Y}| \times d_N}$

Example: The first layer lives in $\mathbb{R}^{d_2 \times d_1}$

$$\mathbf{h}^{(N)} = \sigma^{(N)}(W^{(N)} \cdots \sigma^{(2)}(W^{(2)}\sigma^{(1)}(W^{(1)}\mathbf{e}(\mathbf{x}))))$$

Example: the activation function $\sigma^{(i)}$ in each layer is a design choice, eg. "sigmoid", "relu", "tanh" or "identity" in final layer

The vector $\mathbf{e}(\mathbf{x}) \in \mathbb{R}^{d_1}$ encodes the input *x*.

The MLP universal approximation theorem states that an MLP with single hidden layer and **a sigmoidal activation function** can represent **any function in the unit cube.**
Feature engineering examples: n-grams, one-hot encoding, bag-of-words, word embeddings, bag-of-embeddings.
Architecture engineering: creating a model capable of learning the shape of a complex decision boundary.

Recurrent Neural Networks

Recurrent neural network:

$$p(y_t \mid y_1, \ldots, y_{t-1}) = \frac{\exp(\boldsymbol{\omega}_{y_t} \cdot \mathbf{h}_t)}{\sum_{y' \in V} \exp(\boldsymbol{\omega}_{y'} \cdot \mathbf{h}_t)}$$

$$\mathbf{h}_t = \mathbf{f}(y_{t-1}, \mathbf{h}_{t-1})$$

Vanishing gradients problem: If the parameter matrix is small, the small values will be compounded as the distance in time steps increases. The derivatives will then shrink.
Exploding gradients problem: The converse is true if the parameter matrix is large, causing the derivatives to explode.

Why RNN: Although we technically could model long histories with the neural n-gram model, the fixed input dimension of the MLP would grow quite large (n-gram assumption → input).

Language Modeling

Definition: In language modeling we fit a probability distribution over all possible instances of natural language sequences drawn from a vocabulary.
Locally normalized: The distribution over each transition should be a valid probability distribution.
Globally normalized: The scores of trees should be normalized by the sum of weights of all the possible trees under the grammar.

$$Z = \sum_{\mathbf{y}' \in \mathcal{Y}} \prod_{t=1}^{|\mathbf{y}'|} \theta_{\mathbf{y}'_{\le t}}$$

***n*-gram:** each token only depends on a finite history.

$$p(y_t \mid \mathbf{y}_{<t}) \overset{\text{def}}{=} p(y_t \mid y_{t-1}, \ldots, y_{t-n+1})$$

$$\mathbf{h}_t = \mathbf{f}(\mathbf{e}(\text{history})) = \mathbf{f}([\mathbf{e}(y_{t-1}); \mathbf{e}(y_{t-2}); \mathbf{e}(y_{t-3})])$$

Conditional Random Fields (CRFs)

*A **semiring** is an algebraic structure defined as a 5-tuple* $S = (A, \oplus, \otimes, \bar{0}, \bar{1})$

1. $(A, \oplus, \bar{0})$ *is a commutative monoid.*
2. $(A, \otimes, \bar{1})$ *is a monoid.*
3. $\otimes$ *distributes over* $\oplus$ *: for all* $a, b, c \in A$,

$$(a \oplus b) \otimes c = (a \otimes c) \oplus (b \otimes c)$$
$$c \otimes (a \oplus b) = (c \otimes a) \oplus (c \otimes b)$$

4. $\bar{0}$ *is an annihilator for* $\otimes$ *: for all* $a \in A, \bar{0} \otimes a = a \otimes \bar{0} = \bar{0}$.

A conditional random field is an extension of the logistic regression classifier, i.e., it is a conditional probabilistic model for structured prediction.
CRFs are log-linear models.
Algorithm to determine the most probable tag sequence: Viterbi (O(|Y|^2· T))
The CRF is a softmax. In the limit (T → 0), it becomes the structured perceptron.

Constituency Parsing

A context-free grammar G is a quadruple consisting of:
-A finite set of non-terminal symbols N
-A distinguished start non-terminal S -
-An alphabet of terminal symbols Σ
-A set of production rules R of the form N → α
A grammar is in Chomsky Normal Form (CNF) if the right-hand side of every production rule includes either two non-terminals or a single terminal symbol: N1 → N2 N3 or N → a.

$$p(\mathbf{t}) = \frac{1}{Z} \prod_{r \in \mathbf{t}} \exp\{\text{score}(r)\}$$

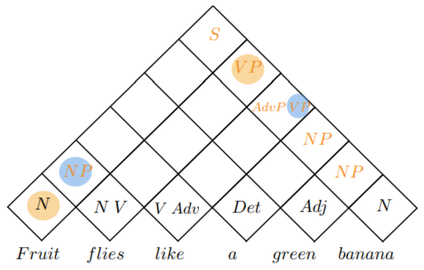


Figure 6.2: Complete CKY chart

The runtime complexity of CKY is O(N^3|R|), where N is the length of the input sequence and |R| is the size of the rule set. The space complexity is O(N^2|N |), where |N | is the size of the set of non-terminals.

Dependency Parsing

We say an edge from node i to node j is projective iff ∀k s.t. i < k < j, node k is a descendant of i. A projective tree is a dependency tree whose edges are all projective, i.e., no edges will cross each other. A tree is non-projective if it does not meet this criterion.

$$L_{ij} = \begin{cases} -A_{ij}, & if\ i \neq j \\ \sum_{k \neq i} A_{kj} & else. \end{cases}$$

With single root constraint:

$$L_{ij} = \begin{cases} -\rho_j, & if\ i = 1 \\ -A_{ij}, & if\ i \neq j \\ \sum_{k \neq i} A_{kj} & else. \end{cases}$$

Matrix Tree Theorem: O(n^3)

$$Z = |L| = \det(L)$$

Lambda Calculus

α–conversion: process of renaming a variable in a lambda term

λx.(x x) y → λt.(t t) y

β–reduction: e process of applying one lambda term to another

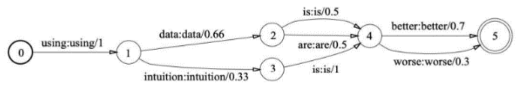
λy.(z ((λx.x z) y)) → λy.(z (z y))

Example:

- λx.λy.(x ((λx.x x) y))
- ∃y. LIKES(ALEX, y) ∧ TEACHER(y), which means that Alex likes some teacher.

Weighted Finite-State Transducers

A weighted finite-state transducer (WFST) is a computationally tractable generalization of a FST, in which each accepting path is assigned a score, computed from the transitions, the initial state, and the final state.



The Floyd–Warshall (FW) algorithm is a dynamic program for finding the transitive closure of a graph G with N vertices in a run time of O(N^3). Put more simply, it is a shortest path-finding algorithm which operates on directed weighted graphs with positive or negative edge weights.

Be able to design basic finite-state models by hand, e.g. *n*-grams models and modes that accept other simple sets of strings.

Sequence-to-Sequence Models

Sequence-to-sequence models typically consist of an encoder and a decoder and map sequences from one domain X into probability distributions over sequences of another domain Y.

z = encoder(x),

x ∈ X p(y | x) = decoder(z), y ∈ Y

The Attention Mechanism enables a model to "attend" to information from different time steps by taking a convex combination of a model's states to build the context vector. It can be formalized as:

c = softmax(score(q, K) ∗ V

Axes of Modeling

Probabilistic models: These models represent conditional distributions over Y. To train a classifier, one uses the dataset to compute p(y | x), for x, y ∈ X × Y. A classification rule based on these conditional probabilities is then used to choose a label for y.

Non-probabilistic models: These models typically operate by learning rules to separate the feature space. The model then returns the class associated with the space where it believes a sample comes from.

Generative: Such approaches fit a distribution p over X × Y. For use in classification tasks, we then compute the conditional distribution p(y | x), for x, y ∈ X × Y by Bayes' rule (n-gram models, Markov random fields, and some recurrent neural networks).

Discriminative: Such approaches skip the fitting of p and directly fit a distribution p(y | x) (logistic regression, conditional random fields, and some recurrent neural networks).

Regularization: It is important that any fitted model generalizes well, i.e., correctly predicts labels on unseen data. To this end, we must avoid overfitting the model to our training data, which we measure as the model's generalization error.

EXTRAS:

The gradient log-linear model:

$$\mathrm{LL}(\boldsymbol{\theta}) := -\sum_{i=1}^{n} \log(p(y_i \mid \mathbf{x}_i, \boldsymbol{\theta}))$$

$$p(y \mid \mathbf{x}, \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y))}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y'))}$$

$$\begin{aligned} \frac{\partial LL(\boldsymbol{\theta})}{\partial \theta_k} &= -\sum_{i=1}^{n} \frac{\partial \log(p(y_i \mid \mathbf{x}_i, \boldsymbol{\theta}))}{\partial \theta_k} \\ &= -\sum_{i=1}^{n} \frac{\partial}{\partial \theta_k} \left(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y) - \log \left(\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y')) \right) \right) \\ &= -\sum_{i=1}^{n} \left(f_k(\mathbf{x}_i, y_i) - \frac{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}_i, y')) f_k(\mathbf{x}_i, y')}{\sum_{y' \in \mathcal{Y}} \exp(\boldsymbol{\theta} \cdot \mathbf{f}(\mathbf{x}, y'))} \right) \\ &= -\sum_{i=1}^{n} \left(f_k(\mathbf{x}_i, y_i) - \sum_{y' \in \mathcal{Y}} p(y' \mid \mathbf{x}_i, \boldsymbol{\theta}) f_k(\mathbf{x}_i, y') \right) \\ &= -\sum_{i=1}^{n} f_k(\mathbf{x}_i, y_i) + \sum_{i=1}^{n} \sum_{y' \in \mathcal{Y}} p(y' \mid \mathbf{x}_i, \boldsymbol{\theta}) f_k(\mathbf{x}_i, y') \\ &= -\sum_{i=1}^{n} f_k(\mathbf{x}_i, y_i) + \sum_{i=1}^{n} \mathbb{E}_{Y \mid X = x_i} [f_k(\mathbf{x}_i, Y)] \end{aligned}$$

Boolean semiring: ({0, 1}, ∨, ∧, 0, 1)
Probability semiring is: (R, +, ×, 0, 1)

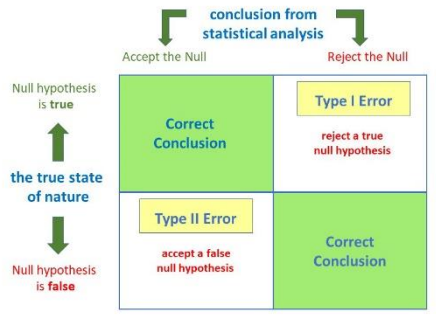
Computational complexity CRFS (n): O(NE(C^nT + C^nK)) with C = |Y|. This is because we compute the gradient NE (samples × epochs) times, one computation of the gradient is of complexity C^nT +C^nK due to forward-backward on trigrams and the computation of C^n possible scores by inner products of K-dimensional vectors. The complexity of the feature function can be ignored.

```
$\gamma(\mathbf{w}, t_N) \leftarrow 1$
for $n \leftarrow N-1, \ldots, 0$:
    $\gamma(\mathbf{w}, t_n) \leftarrow \boxed{\max_{t_{n+1} \in \mathcal{T}}} \exp\{\mathrm{score}(\langle t_n, t_{n+1} \rangle, \mathbf{w})\} \times \gamma(\mathbf{w}, t_{n+1})$
```

McNemar's test: compare classifiers → chi-squared distribution

Permutation test: provide a simple method for constructing the sampling distribution of a test statistic through empirical observations + allow us to test whether a classifier performs better than random chance

5x2cv paired t-test: compare performance of two classifiers based on some metric p → t distribution



conclusion from statistical analysis
Accept the Null
Reject the Null
Null hypothesis is true
the true state of nature
Null hypothesis is false
Correct Conclusion
Type I Error
reject a true null hypothesis
Type II Error
accept a false null hypothesis
Correct Conclusion